Naloga se nanaša na nalogi prejšnjih dveh tednov (Šikaniranje, Lokacije ovir): večinoma je potrebno le preoblikovati to, kar ste programirali takrat, v funkcije. Pri reševanju smete uporabiti tudi objavljene rešitve prejšnjih dveh nalog.

Napisati je potrebno naslednje funkcije:

- `stevilo_ovir(ovire)` prejme ovire v obliki seznama trojk `(x0, x1, y)` in vrne število ovir.
- `dolzina_ovir(ovire)` vrne skupno dolžino vseh ovir
- `sirina(ovire)` vrne širino kolesarke steze, se pravi najbolj desno koordinato v seznamu ovir
- `pretvori_vrstico(vrstica)` prejme vrstico v obliki niza, sestavljenega iz znakov `#` in `.` ter vrne seznam parov (x0, x1) (ogrevalni del domače naloge prejšnjega tedna)
- `dodaj_vrstico(bloki, y)` prejme seznam, kakršnega vrača prejšnja funkcija in neko število `y`. Vrniti mora seznam, v katerem je vsakemu paru dodan `y`. Klic `dodaj_vrstico([(3, 4), (6, 8), (11, 11)], 3)` vrne `[(3, 4, 3), (6, 8, 3), (11, 11, 3)]`.
- `pretvori_zemljevid(zemljevid)` dobi zemljevid v obliki seznama nizov in vrne seznam ovir (obvezna domača naloga prejšnjega tedna). Funkcija mora uporabiti prejšnji dve funkciji - testi bodo to preverili tako, da bodo začasno zamenjali tvojo funkcijo z neko drugo, ki vrača napačne številke in preverili, ali so številke res "pravilno napačne".
- `globina(ovire, x)` prejme seznam ovir (v obliki trojk) in vrne vrstico, v kateri bi kolesar, ki se vozi po stolpcu `x`, naletel na oviro (naloga izpred dveh tednov).
- `naj_stolpec(ovire)` vrne stolpec, v katerem kolesar pride najdlje in vrstico, do katere pride. Možno je tudi, da v kakem stolpcu ni ovire; v tem primeru vrne koordinato tega stolpca in `None`.
- `senca(ovire)` vrne seznam, katerega elementa so `True` oz. `False` glede na to, ali stolpec vsebuje kako oviro ali ne. Če je širina poti 5 in sta drugi in zadnji stolpec brez ovir, vrne `[True, False, True, True, False]`.

Rešitev

```
def stevilo_ovir(ovire):
    return len(ovire)

def dolzina_ovir(ovire):
    dolzina = 0
    for x0, x1, _ in ovire:
        dolzina += x1 - x0 + 1
    return dolzina

def sirina(ovire):
    najx = 0
    for _, x1, _ in ovire:
```

```
        if x1 > najx:
            najx = x1
    return najx

def pretvori_vrstico(vrstica):
    vrstica = "." + vrstica + "."
    bloki = []
    for i, znak in enumerate(vrstica):
        if znak == "#":
            if vrstica[i - 1] == ".":
                zacetek = i
            if vrstica[i + 1] == ".":
                bloki.append((zacetek, i))
    return bloki

def dodaj_vrstico(bloki, y):
    z_vrstico = []
    for x0, x1 in bloki:
        z_vrstico.append((x0, x1, y))
    return z_vrstico

def pretvori_zemljevid(zemljevid):
    ovire = []
    for y, vrstica in enumerate(zemljevid, start=1):
        ovire += dodaj_vrstico(pretvori_vrstico(vrstica), y)
    return ovire

def globina(ovire, x):
    najy = None
    for x0, x1, y in ovire:
        if x0 <= x <= x1 and (najy == None or y < najy):
            najy = y
    return najy

def naj_stolpec(ovire):
    naj_y = 0
    for x in range(1, sirina(ovire) + 1):
        g = globina(ovire, x)
        if g == None:
            return x, None
        if g > naj_y:
            naj_y = g
            naj_x = x
    return naj_x, naj_y

def senca(ovire):
```

```
    zaprti = []
    for x in range(1, sirina(ovire) + 1):
        zaprti.append(globina(ovire, x) == None)
    return zaprti
```

Komentar si zasluži le par funkcij.

pretvori_zemljevid V `pretvori_zemljevid` dodamo vrstico z `+=` ne z `append`.

```
# Tole ne deluje!
    for y, vrstica in enumerate(zemljevid, start=1):
        ovire.append(dodaj_vrstico(pretvori_vrstico(vrstica), y))
```

Metoda `append` bi dodala vrstico kot nov element seznama - se pravi vrstico, ne njenih elementov. Tako bi dobili seznam seznamov.

Alternative so `extend`, ki naredi isto kot `+=`,

```
    for y, vrstica in enumerate(zemljevid, start=1):
        ovire.extend(dodaj_vrstico(pretvori_vrstico(vrstica), y))
```

ali pa dodajanje posamičnih elementov, kar pa je daljše, nepotrebno in počasnejše

```
    for y, vrstica in enumerate(zemljevid, start=1):
        for ovira in dodaj_vrstico(pretvori_vrstico(vrstica), y):
            ovire.append(ovira)
```

V tej funkciji je zanimivo tudi, kako pridemo do številke vrstice. Lahko štejemo sami, s spremenljivko, ki jo sami povečujemo, lahko pa uporabimo `enumerate`. Če funkciji `enumerate` dodamo argument `start=1`, bo štela od `1`. Sicer pa pač šteje od `0` in kot številko vrstice uporabimo `y + 1`.

naj_stolpec V tej funkciji je praktično, da takrat, ko najdemo prost stolpec, kar takoj vrnemo `x, None`. To prekine zanko, konča funkcijo, vrne rezultat.

senca Dve stvari. Ta funkcija lahko kliče funkcijo `globina`. Pravzaprav je zelo koristno, da jo, saj sicer ponovimo večino kode te funkcije.

Druga:

```
zaprti.append(globina(ovire, x) == None)
```

in ne

```
if globina(ovire, x) == None:
    zaprti.append(True)
else:
    zaprti.append(False)
```

Seveda deluje tudi drugo, vendar ni smiselno, saj je `globina(ovire, x) == None` že vrednost, ki jo želimo dodati v seznam. (Zanimivo: študenti rezultat

primerjanja stalno dojemajo kot *pogoj*, ki ga lahko uporabljajo v `if` in `while`, ne pa kot vrednost, ki jo lahko uporabijo, kjerkoli. Mogoče bi se to rešilo tako, da bi že eno predavanje, preden začnemo s pogoji, predstavil logične izraze, kot nekaj, kar pač vrne `True` ali `False`?!)